Format JSON

> Version PDF du document http://tvaira.free.fr/projets/activites/activite-json.html
> Thierry Vaira tvaira@free.fr
> 2019 (rev. 1.0)

Notions de base

Wikipédia :

> JSON (*JavaScript Object Notation*) est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée comme le permet XML par exemple. Créé par Douglas Crockford entre 2002 et 2005, il est décrit par la RFC 7159 de l'IETF.

Par définition, JSON est un format d'échange de données.

Site web : json.org

Un document JSON ne comprend que deux types d'éléments structurels :

- des ensembles de paires « nom » (alias « clé ») / « valeur » : `"id": "file"`
- des listes de valeurs séparées par des virgules : `"value": "New", "onclick": "CreateNewDoc()"`.

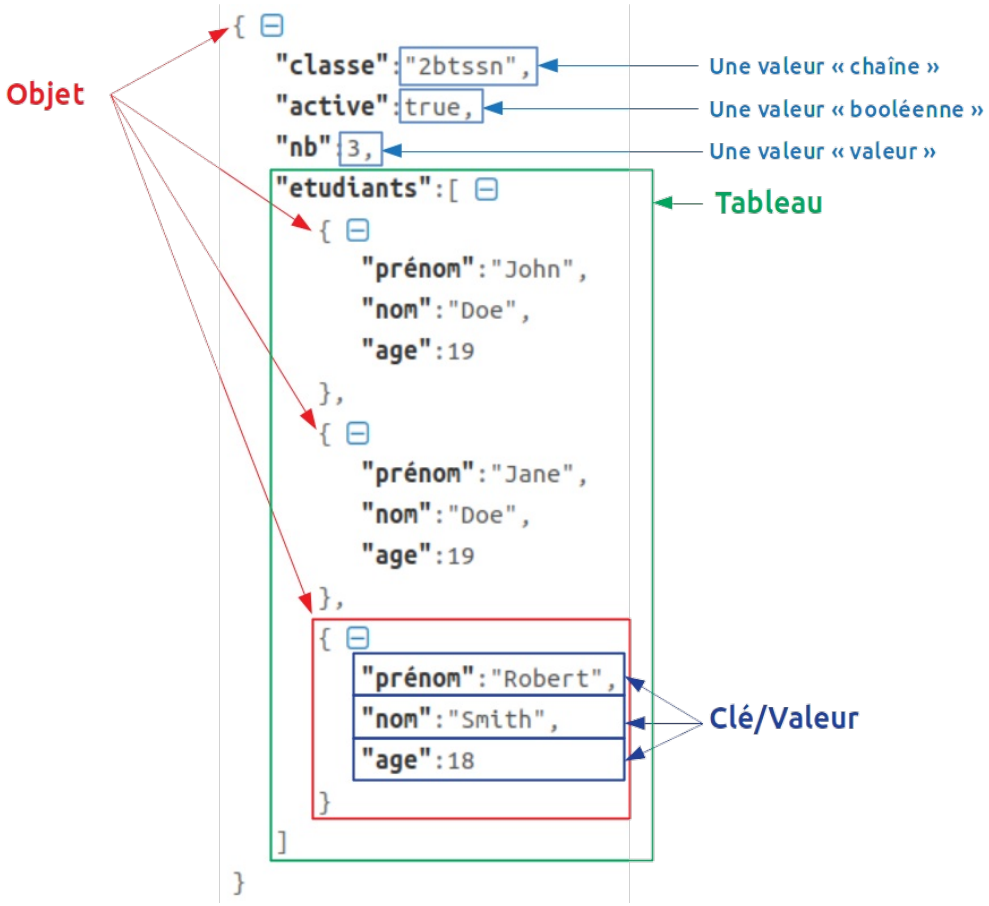
Ces mêmes éléments représentent trois types de données :

- des objets : `{ ... }` ;
- des tableaux : `[ ... ]` ;
- des valeurs génériques de type tableau, objet, booléen, nombre, chaîne de caractères ou *null* (valeur vide).

Exemple de format JSON :

```
{
    "menu": {
        "id": "file",
        "value": "File",
        "popup": {
            "menuitem": [
                { "value": "New", "onclick": "CreateNewDoc()" },
                { "value": "Open", "onclick": "OpenDoc()" },
                { "value": "Close", "onclick": "CloseDoc()" }
            ]
        }
    }
}
```

Remarque : l'extension des fichiers au format JSON est `.json`.



Même si le format est simple, il est possible d'utiliser un validateur en ligne.

Site web : json.org

Avantages

On peut citer :

- peu verbeux, ce qui le rend lisible aussi bien par un humain que par une machine ;
- facile à apprendre, car sa syntaxe est réduite et non extensible (bien qu'il souffre de quelques limitations) ;
- ses types de données sont connus et simples à décrire ;
- simple à mettre en œuvre par un développeur

Invénients

Le JSON ne permet pas :

- de représenter d'autres types (format Date, couleurs, …). On utilise donc des représentations sous forme de chaînes de caractères ;
- de sécuriser les données : pas de limite fixe pour les valeurs des entiers, pas de nombres inexistants (*NaN*), aucune distinction entre entier et flottant ;
- les commentaires utiles pour certains types de fichier (configuration par exemple).

Utilisations

Bien qu'utilisant une notation JavaScript, JSON est indépendant du langage de programmation. Étant un format d'échanges de données, JSON sert essentiellement à faire communiquer des applications dans un

environnement hétérogène (AJAX et les services Web par exemple).

Il peut aussi être utilisé pour :

- la sérialisation et désérialisation d'objets ;
- les fichiers de configuration.

Des bibliothèques pour JSON existent dans la plupart des langages de programmation.

En JavaScript, un document JSON représente un objet :

```
var objet = JSON.parse(donnees_json);
```

API Qt

Qt fournit un support du format JSON et possède donc une API C++ pour analyser, modifier et enregistrer des données JSON.

Quelques classes :

- QJsonDocument : lire et d'écrire des documents JSON
- QJsonObject : un objet JSON
- QJsonValue : une valeur en JSON
- QJsonArray : un tableau JSON

Principe

Document JSON

On construit habituellement un document JSON à partir d'une chaîne de caractères au format JSON. Pour cela, on utilise la méthode statique `fromJson()` de la classe `QJsonDocument` en lui passant en paramètre un `QByteArray` contenant les données JSON.

```
// ici un tableau d'objets
QString donnees = "[{"age":19,"nom":"Doe","prénom":"John"},{"age":19,"nom":"Doe","prénom":"Jane"}
,{"age":18,"nom":"Smith","prénom":"Robert"}]";
QJsonDocument documentJSON = QJsonDocument::fromJson(donnees.toUtf8());
```

Il est possible d'obtenir les données JSON à partir d'un fichier :

```
QFile fichierJSON("test.json");
if (fichierJSON.open(QFile::ReadOnly | QFile::Text))
{
    // lecture les données du fichier JSON
    QString donnees = fichierJSON.readAll();

    // création d'un document JSON
    QJsonDocument documentJSON = QJsonDocument::fromJson(donnees.toUtf8());

    //...

    fichierJSON.close();
```

```
}
```

Accéder aux données

Il est possible d'obtenir le contenu du document sous deux formes :

- `QJsonDocument::Indented` : vue indentée lisible par un être humain (choix par défaut)
- `QJsonDocument::Compact` : vue compact (sans mise en forme)

```
qDebug() << Q_FUNC_INFO << documentJSON.toJson();
qDebug() << Q_FUNC_INFO << documentJSON.toJson(QJsonDocument::Compact);
```

On obtient un `QByteArray` encodé en UTF-8 du contenu du document JSON :

```
{
    "classe": "2btssn",
    "active": true,
    "nb": 3,
    "etudiants": [
        { "prénom": "John", "nom": "Doe", "age": 19 },
        { "prénom": "Jane", "nom": "Doe" , "age": 19 },
        { "prénom": "Robert", "nom": "Smith" , "age": 18 }
    ]
}
```

Ou :

```
"{\"active\":true,\"classe\":\"2btssn\",\"etudiants\":[{\"age\":19,\"nom\":\"Doe\",\"pr\xC3\xA9no
m\":\"John\"},{\"age\":19,\"nom\":\"Doe\",\"pr\xC3\xA9nom\":\"Jane\"},{\"age\":18,\"nom\":\"Smith
\",\"pr\xC3\xA9nom\":\"Robert\"}],\"nb\":3}"
```

On va supposer que les données JSON du document contiennent soit un objet (`QJsonObject`) soit un tableau (`QJsonArray`) :

```
if(documentJSON.isObject()) // objet racine ?
{
    QJsonObject objetJSON = documentJSON.object();
    //...
}
else if(documentJSON.isArray()) // tableau racine ?
{
    QJsonArray tableauJSON = documentJSON.array();
    //...
}
```

On va retrouver dans cet objet (ou ce tableau) des valeurs qui peuvent être de type tableau, objet, booléen, nombre, chaîne de caractères ou *null* (valeur vide).

Pour les types `QJsonArray` et `QJsonObject` , le nombre d'éléments contenus peut s'obtenir en appelant les méthodes `count()` ou `size()` .

Les données d'un objet `QJsonObject` sont structurées en un ensemble de clés/valeurs :

```
QStringList listeCles;
listeCles = objetJSON.keys();

// Les clés
qDebug() << listeCles; // une liste de QString

// Les valeurs
for(int i = 0; i < listeCles.count()-1; i++)
{
    qDebug() << objetJSON[listeCles.at(i)]; // un QJsonValue
}
```

Les clés sont des chaînes de caractères de type `QString`. Les valeurs sont de type `QJsonValue`.

```
// 6 types
if(valeur.isArray())
{
   qDebug() << QString("[tableau]");
}
else if(valeur.isBool())
{
   qDebug() << QString::fromUtf8("[booléen]");
}
else if(valeur.isDouble())
{
   qDebug() << QString("[valeur]");
}
else if(valeur.isNull())
{
   qDebug() << QString("[null]");
}
else if(valeur.isObject())
{
   qDebug() << QString("[objet]");
}
else if(valeur.isString())
{
   qDebug() << QString::fromUtf8("[chaîne]");
}
```

Les valeurs de type `QJsonValue` peuvent être converties dans leur type :

```
// 6 types
if(valeur.isArray())
{
   qDebug() << valeur.toArray(); // le type QJsonArray
}
else if(valeur.isBool())
{
   qDebug() << valeur.toBool();
}
else if(valeur.isDouble())
{
   qDebug() << valeur.toDouble();
}
else if(valeur.isObject())
{
```

```
    qDebug() << valeur.toObject(); // le type QJsonObject
}
else if(valeur.isString())
{
    qDebug() << valeur.toString();
}
```

Pour un `QJsonObject`, on pourra accèder directement à sa valeur en utilisant : la méthode `value()` ou l'opérateur `[]` en indiquant la clé. Dans ces deux situations, on récupère un `QJsonValue`.

```
qDebug() << objetJSON.value(QString("nom"));  // Pour obtenir la valeur de la clé "nom"

qDebug() << objetJSON["nom"];  // Pour obtenir la valeur de la clé "nom"
```

Remarque : la méthode `value()` est *const* ce qui signifie qu'elle retourne une copie de la valeur contenu dans l'objet ou `QJsonValue::Undefined` si la clé n'existe pas. Par contre, l'opérateur `[]` retourne un référence modifiable sur la valeur si l'objet n'est pas *const*.

Pour un `QJsonArray`, on pourra accèder directement à sa valeur en utilisant : la méthode `at()` ou l'opérateur `[]` en indiquant un indice. Dans ces deux situations, on récupère un `QJsonValue`.

```
qDebug() << tableauJSON.at(0);  // Pour obtenir la première valeur du tableau

qDebug() << tableauJSON[0];  // Pour obtenir la première valeur du tableau
```

Remarque : la méthode `at()` est *const* ce qui signifie qu'elle retourne une copie de la valeur contenu dans l'objet ou `QJsonValue::Undefined` si l'indice est hors limite. Par contre, l'opérateur `[]` retourne un référence modifiable sur la valeur si l'objet n'est pas *const*.

Insérer des données

Pour les éléments de type `QJsonObject` ou `QJsonArray`, on utilisera leur méthode `insert()`. Pour créer une valeur `QJsonValue`, on utilisera la méthode statique `fromVariant()`.

Exemple pour un objet racine d'un document JSON :

```
QJsonObject objetJSON = documentJSON.object();

// une clé/valeur de type chaine
objetJSON.insert("chaine", QJsonValue::fromVariant("toto"));

// une clé/valeur de type valeur
objetJSON.insert("age", QJsonValue::fromVariant(10));

// une clé/valeur de type tableau
QJsonArray fruits;
fruits.insert(0, QJsonValue::fromVariant("banane"));
fruits.insert(1, QJsonValue::fromVariant("pomme"));
fruits.insert(2, QJsonValue::fromVariant("orange"));
objetJSON.insert("fruits", fruits);

documentJSON.setObject(objetJSON);
```

Supprimer des données

Pour un `QJsonObject`, on pourra supprimer une valeur à partir de sa clé avec `remove()`.

Exemple :

```
QJsonObject objetJSON = documentJSON.object();
QStringList listeCles = objetJSON.keys();

QString cle = "nb"; // nb est la clé de la valeur à supprimer
if(objetJSON.contains(cle))
{
   objetJSON.remove(cle);
   documentJSON.setObject(objetJSON);
}
```

Remarque : Pour un `QJsonArray`, on pourra supprimer une valeur à partir de sa clé avec : `removeAt()` (voir aussi `removeFirst()` / `pop_front()` ou `removeLast()` / `pop_back()`). La méthode `replace()` permet de remplacer un `QJsonValue`.

Sauvegarder un document JSON

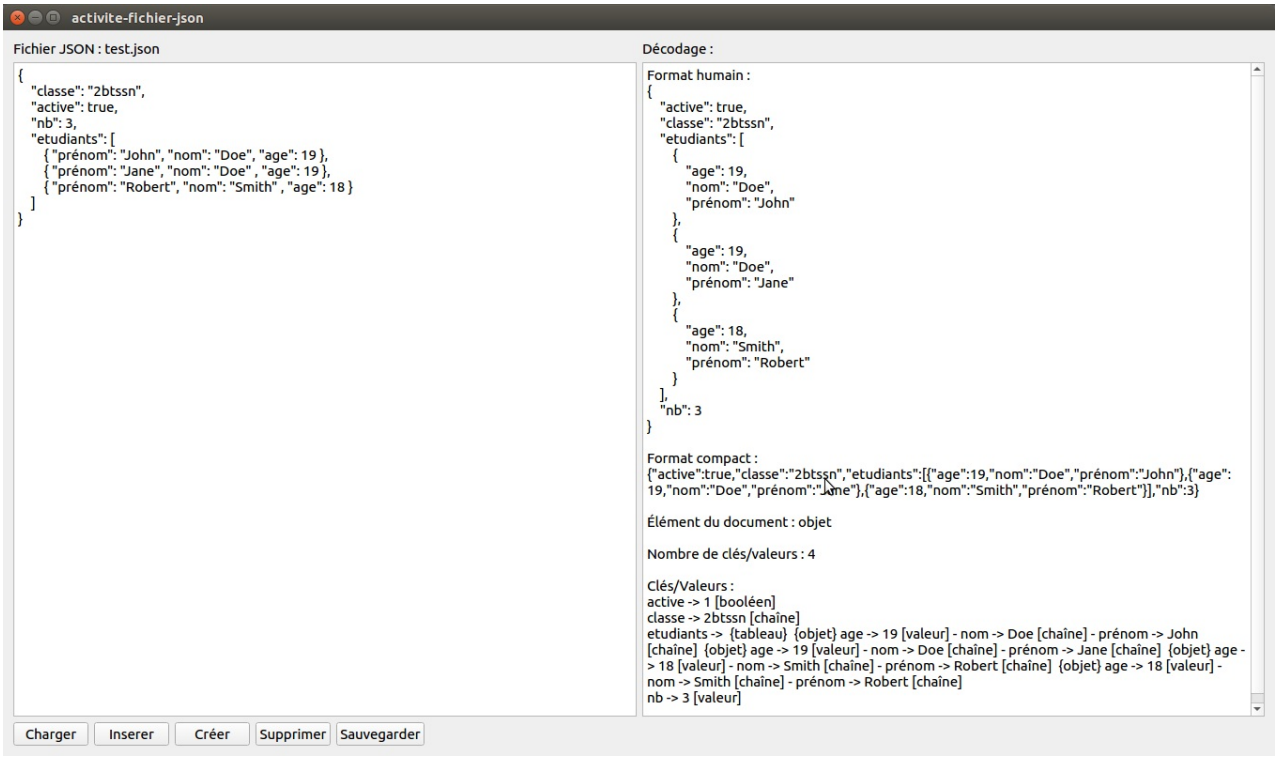
On utilisera un objet `QFile` et sa méthode `write()` à laquelle on passera en paramètre le contenu du document JSON en appelant `toJson()`.

```
QFile fichierJSON("test.json");
QFile fichierJSON(fichier);

if (fichierJSON.open(QFile::WriteOnly | QIODevice::Text))
{
   fichierJSON.write(documentJSON.toJson());
   fichierJSON.close();
}
else
{
   QMessageBox::critical(this, "Erreur", QString::fromUtf8("Erreur enregistrement !"), QMessageBo
x::Ok, 0);
}
```

Code source

Exemple : test-mo-fichier-json.zip



Remarque : Les clés/valeurs ne sont pas ordonnées dans un document JSON : cela signifie que l'ordre n'est pas important. Qt ordonne les clés alphabétiquement (à cause de `QMap`) mais cela n'aura pas de répercussion pour JSON. Si l'ordre est important pour vous, il vous faudra faire des objets avec une seule paire clé/valeur dans un tableau par exemple.

Voir aussi

- JSON Save Game Example qui montre notamment la sérialisation d'un objet en JSON